Éléments d'informatique – Cours 11. Pile d'appel, fonctions récursives.

Pierre Boudes

6 décembre 2010

This work is licensed under the *Creative Commons Attribution-NonCommercial-ShareAlike 3.0 License*.

Pile d'appel

- Rappel sur les fonctions en C
- Trace et mémoire
- Pile d'appel

Fonctions récursives

- Définition et analogie mathématique
- Exemple de la factorielle
- Pour aller plus loin
- Exemples

Rappel sur les fonctions en C

Utilisation des fonctions :

- *déclaration* (types des paramètres et de la valeur de retour)
- *définition* (code, paramètres formels)
- *appel* (paramètres effectifs, **espace mémoire**)

Rappel sur les fonctions en C

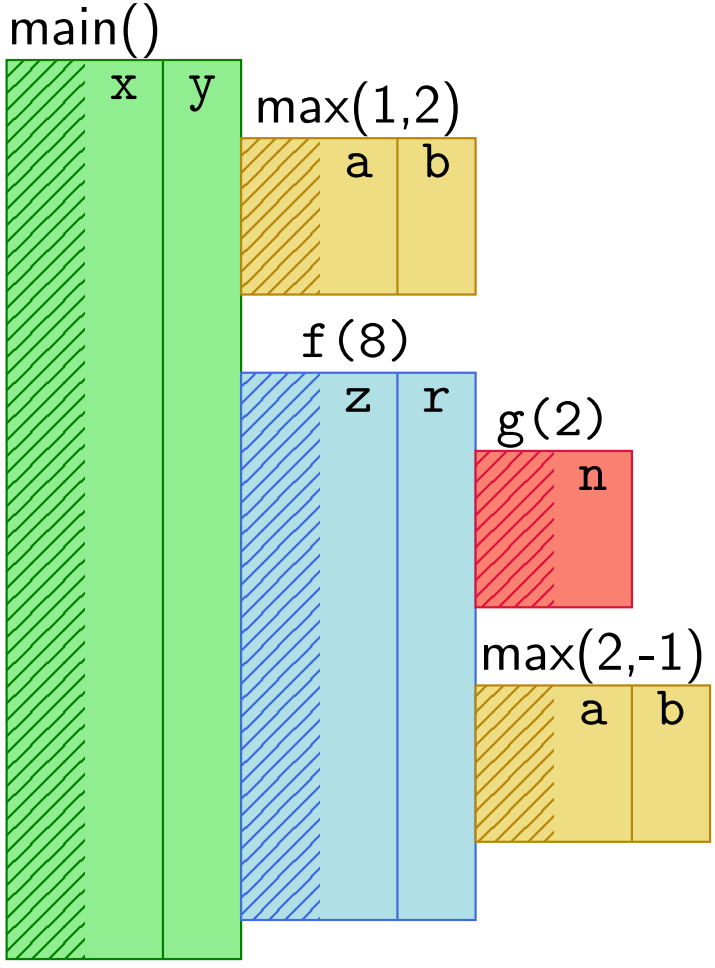
Utilisation des fonctions :

- *déclaration* (types des paramètres et de la valeur de retour)
- *définition* (code, paramètres formels)
- *appel* (paramètres effectifs, **espace mémoire**)

Voyons de manière plus précise cette question d'espace mémoire.

Trace et mémoire

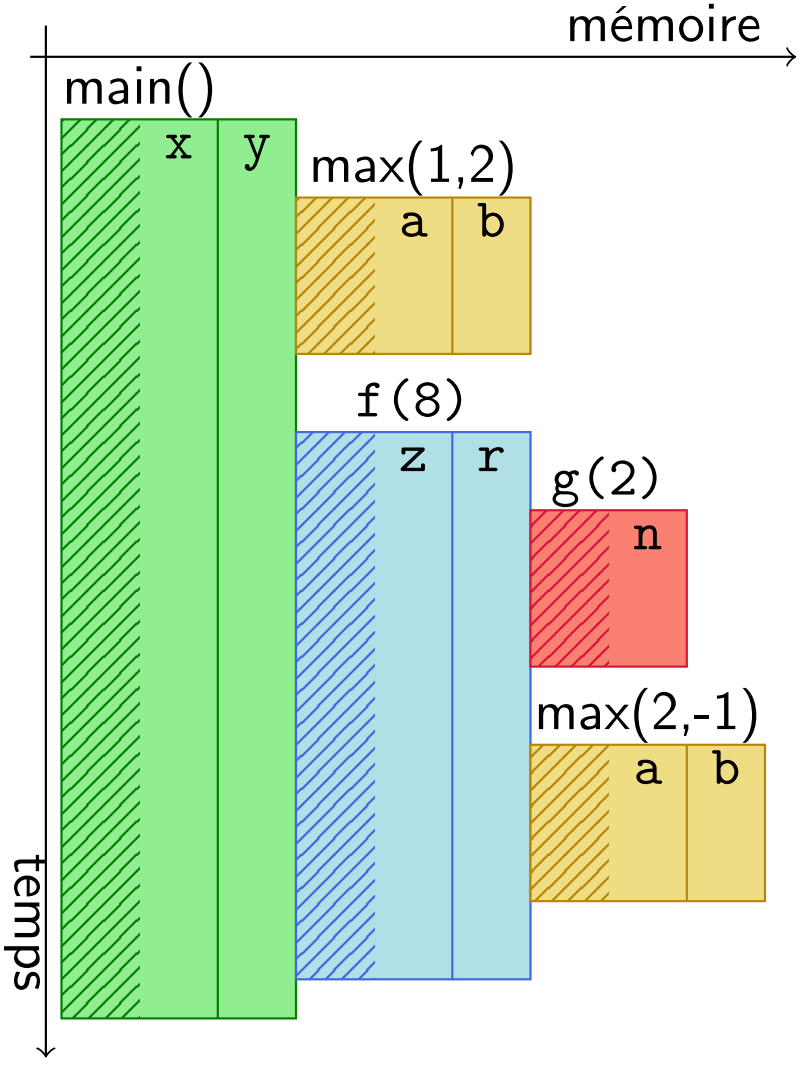
La trace d'un programme avec fonctions donne schématiquement ce type de dessin



Trace et mémoire

La trace d'un programme avec fonctions donne schématiquement ce type de dessin

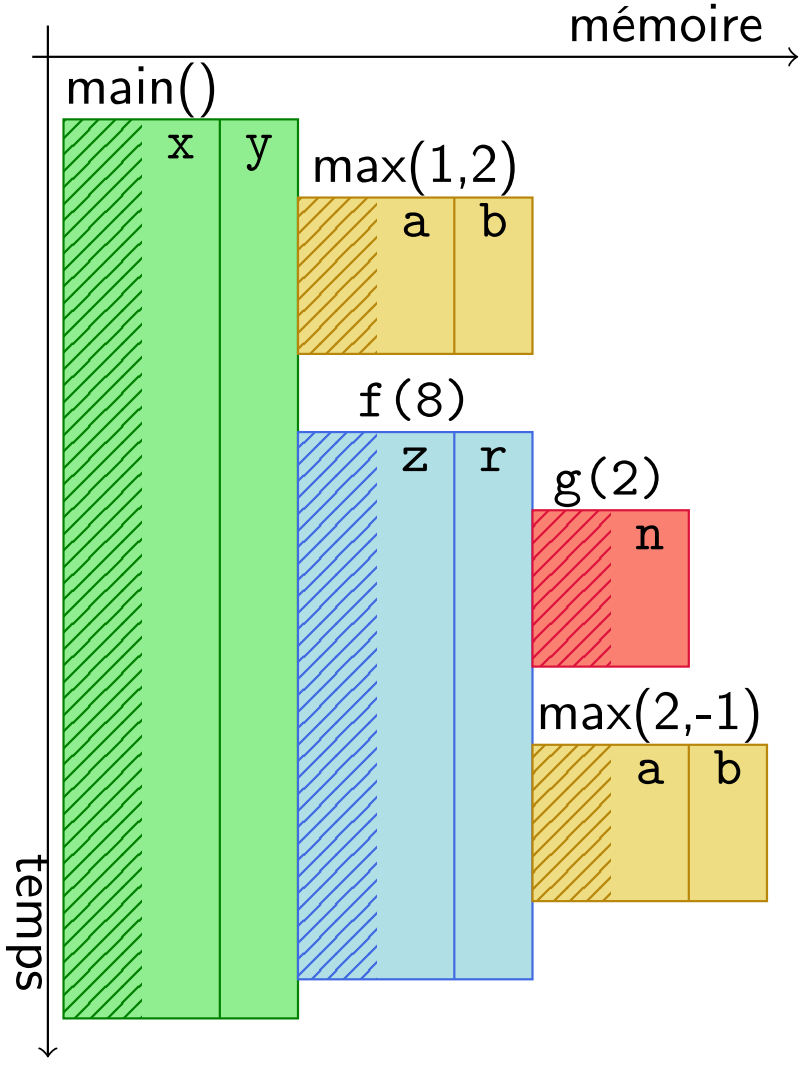
- verticalement c'est le temps
- et horizontalement l'occupation mémoire



Trace et mémoire

La trace d'un programme avec fonctions donne schématiquement ce type de dessin

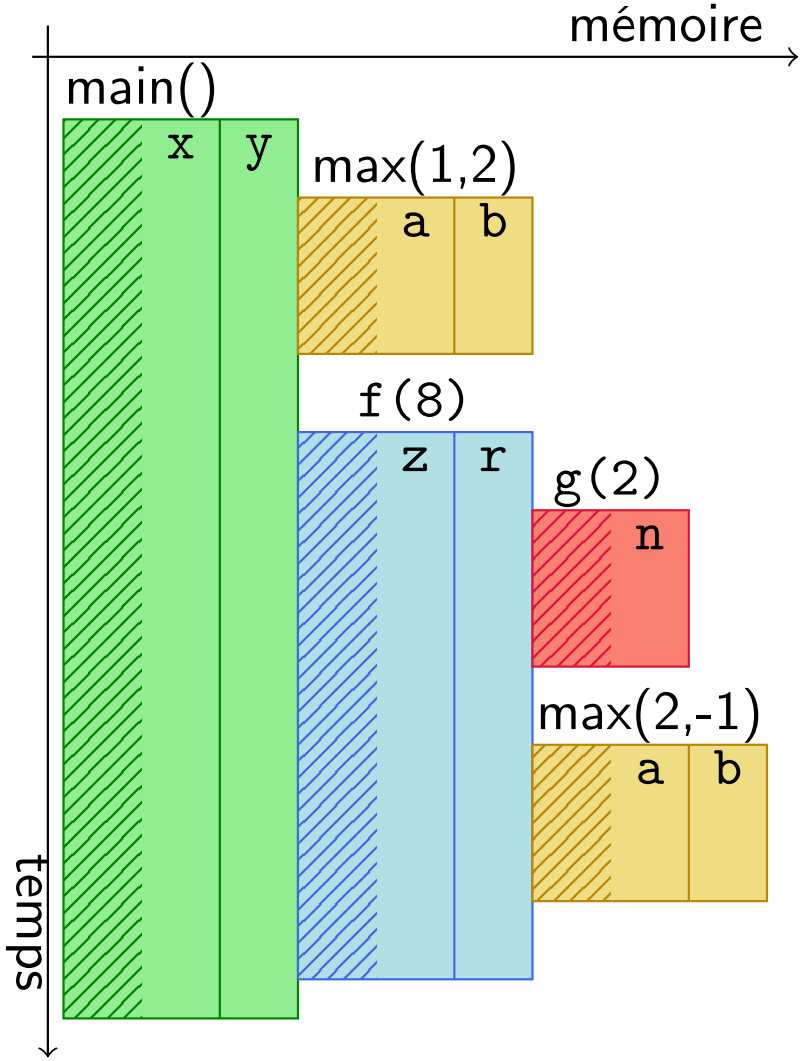
- verticalement c'est le temps
- et horizontalement l'occupation mémoire
- un appel de fonction occupe une portion de mémoire, puis la libère



Trace et mémoire

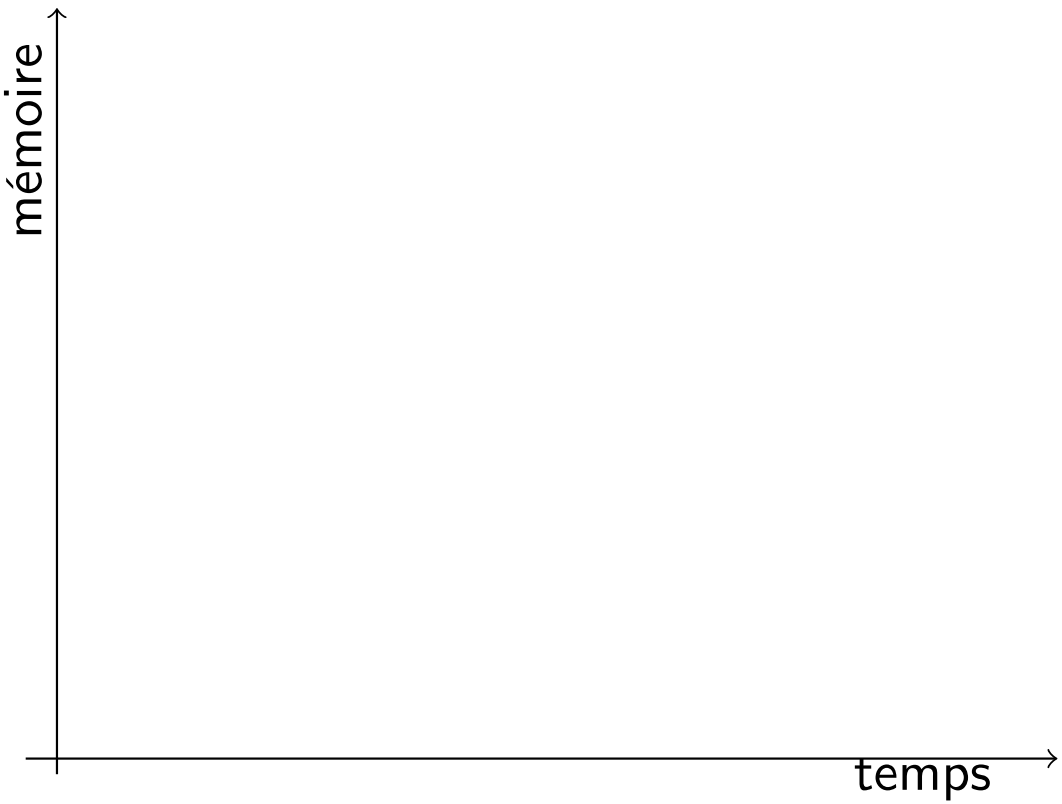
La trace d'un programme avec fonctions donne schématiquement ce type de dessin

- verticalement c'est le temps
- et horizontalement l'occupation mémoire
- un appel de fonction occupe une portion de mémoire, puis la libère
- c'est réellement ce qui arrive dans la mémoire de vos programmes



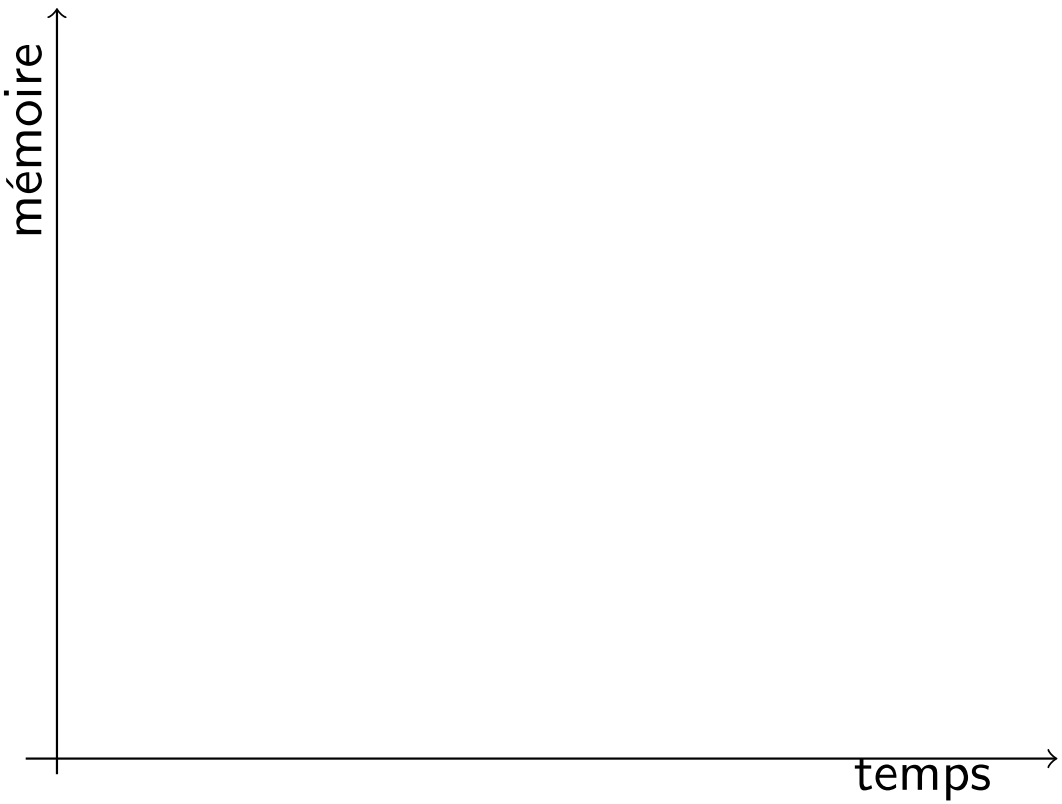
Pile d'appel

On parle de pile d'appel car les appels de fonctions s'empilent...



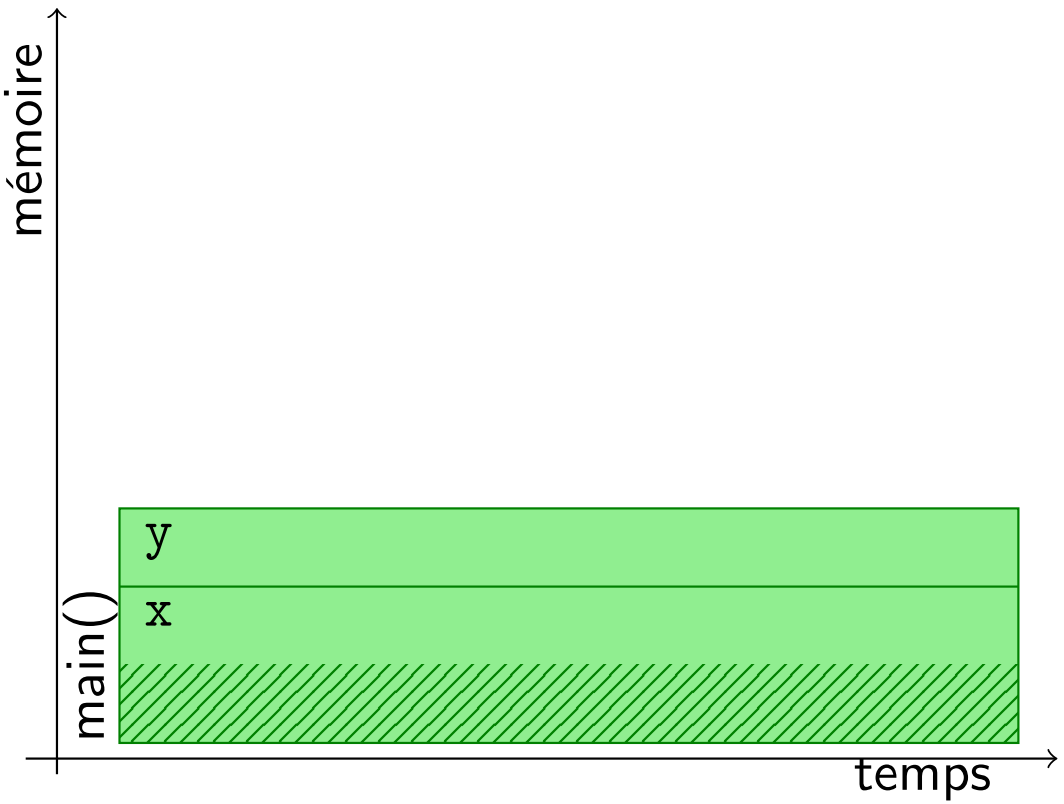
Pile d'appel

On parle de pile d'appel car les appels de fonctions s'empilent...
comme sur une une pile d'assiettes !



Pile d'appel

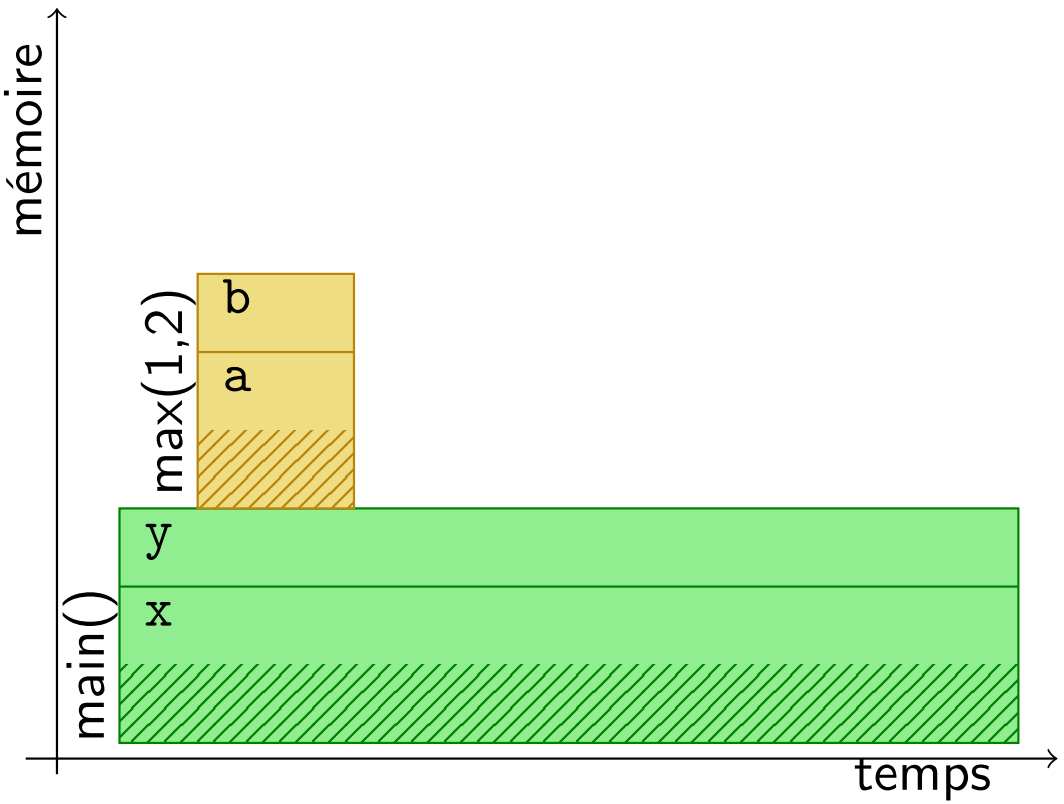
On parle de pile d'appel car les appels de fonctions s'empilent...
comme sur une une pile d'assiettes !



Pile d'appel

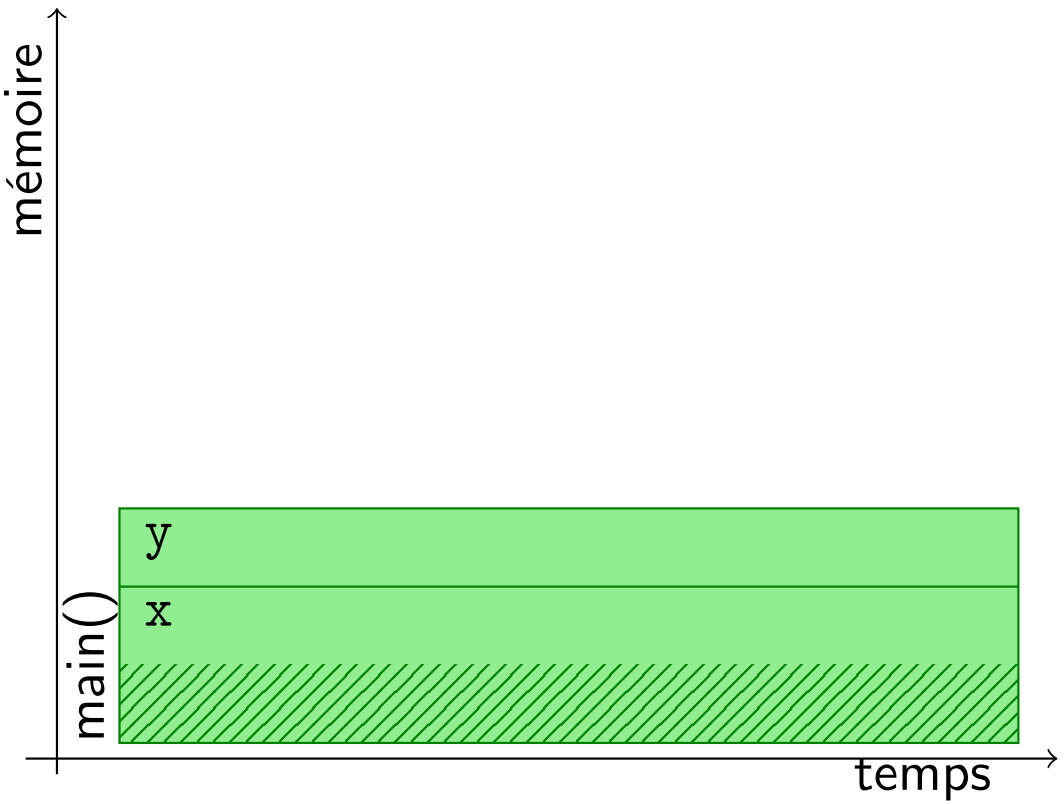
On parle de pile d'appel car les appels de fonctions s'empilent...

comme sur une une pile d'assiettes !



Pile d'appel

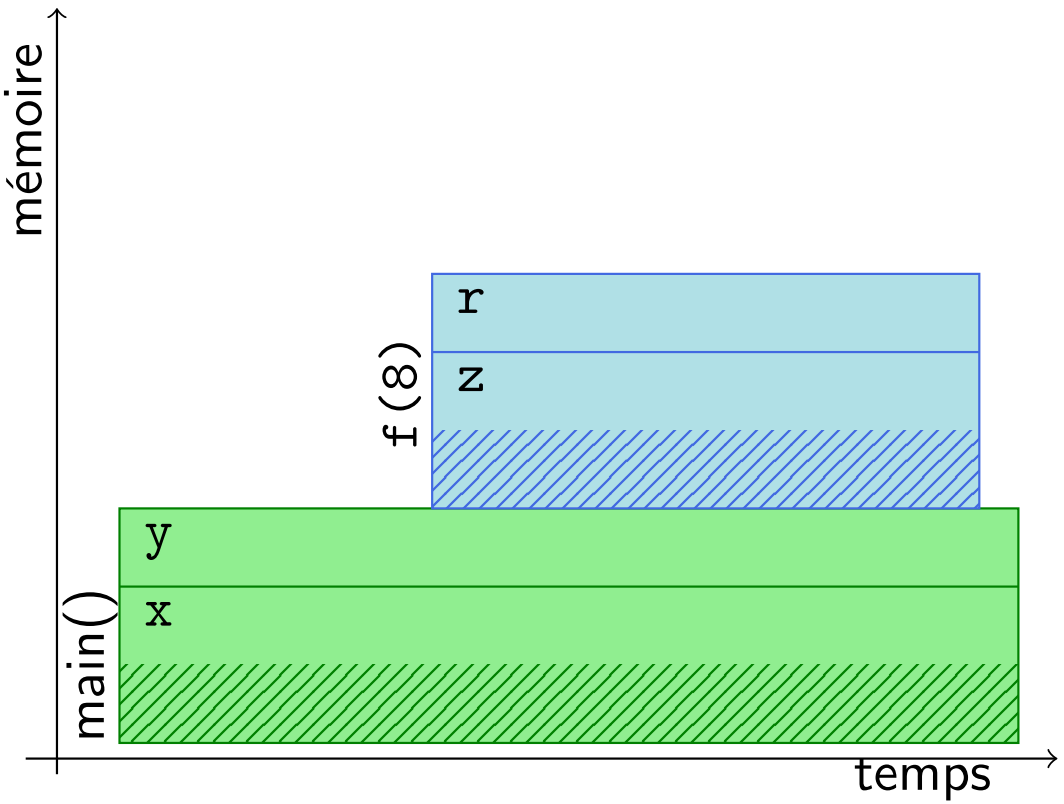
On parle de pile d'appel car les appels de fonctions s'empilent...
comme sur une une pile d'assiettes !



Pile d'appel

On parle de pile d'appel car les appels de fonctions s'empilent...

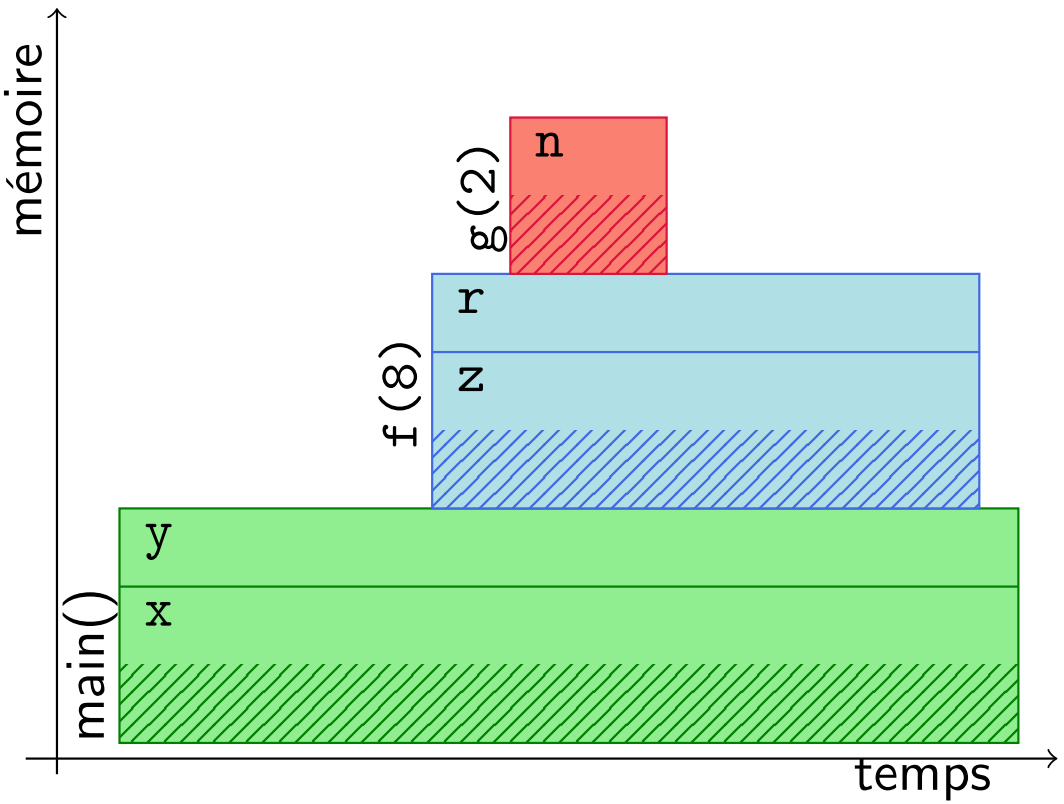
comme sur une une pile d'assiettes !



Pile d'appel

On parle de pile d'appel car les appels de fonctions s'empilent...

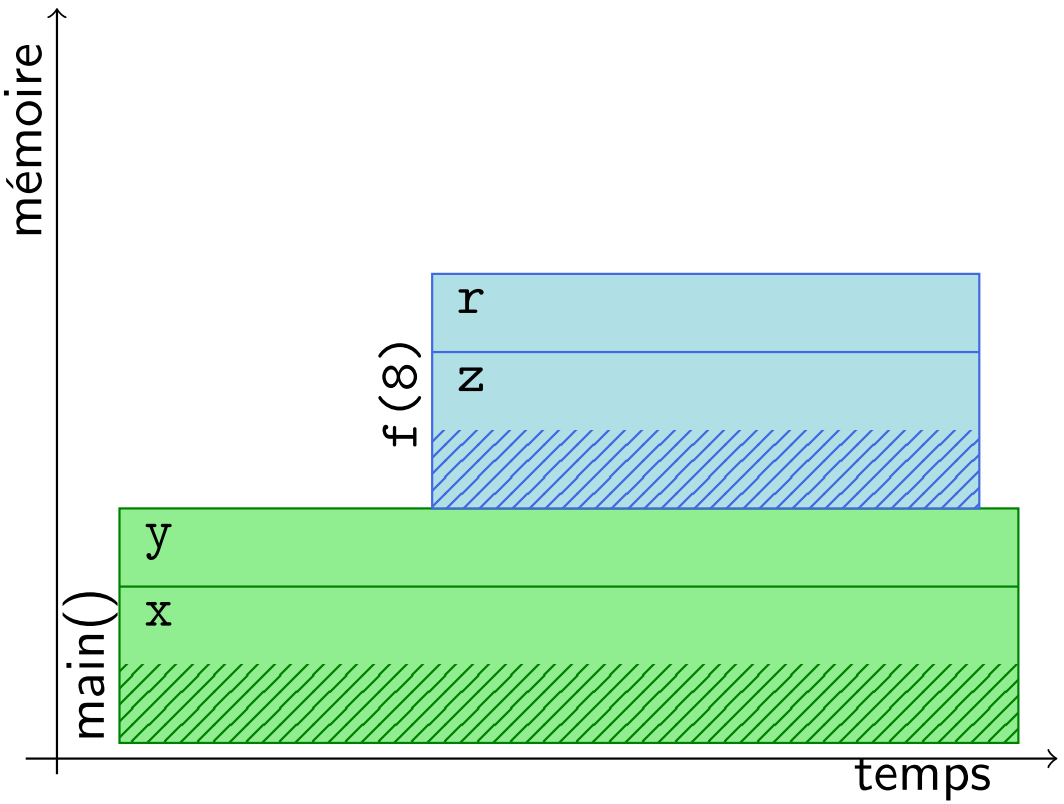
comme sur une une pile d'assiettes !



Pile d'appel

On parle de pile d'appel car les appels de fonctions s'empilent...

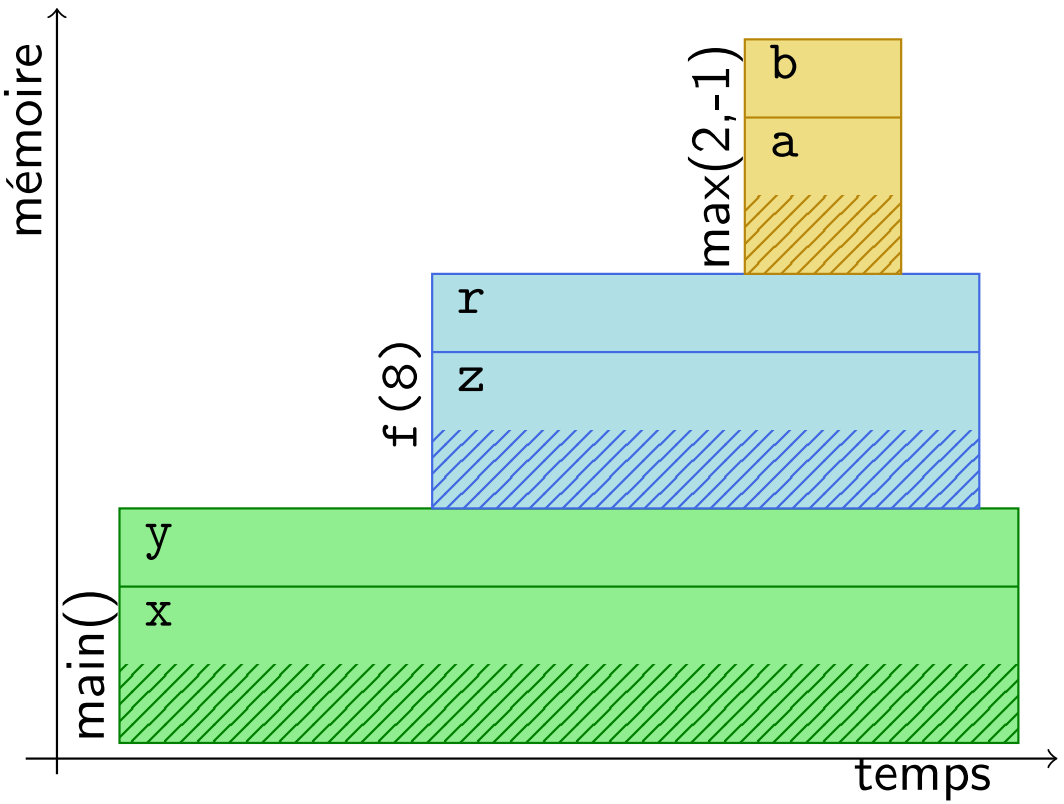
comme sur une une pile d'assiettes !



Pile d'appel

On parle de pile d'appel car les appels de fonctions s'empilent...

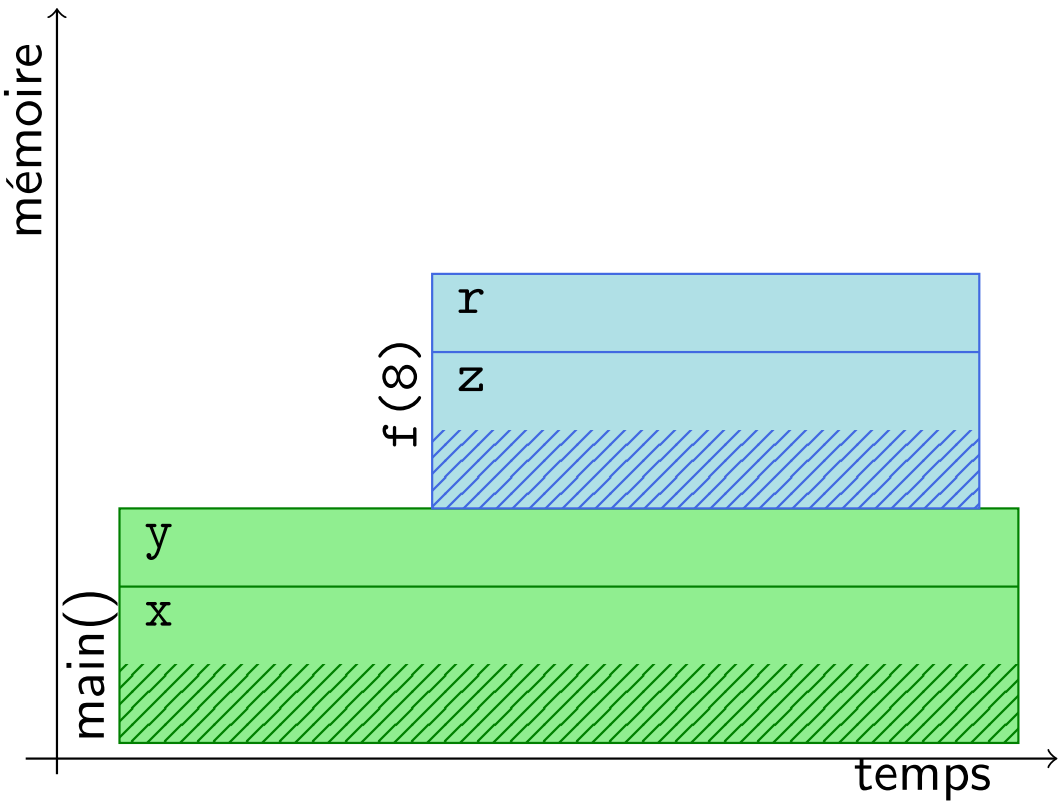
comme sur une une pile d'assiettes !



Pile d'appel

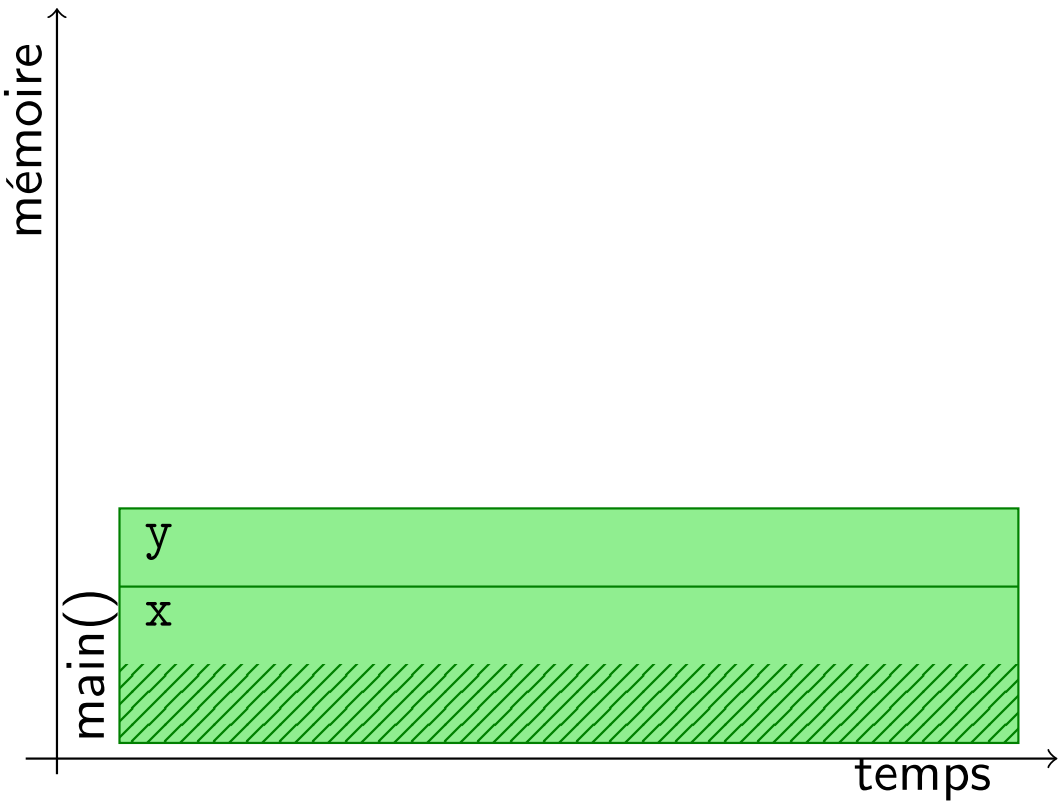
On parle de pile d'appel car les appels de fonctions s'empilent...

comme sur une une pile d'assiettes !



Pile d'appel

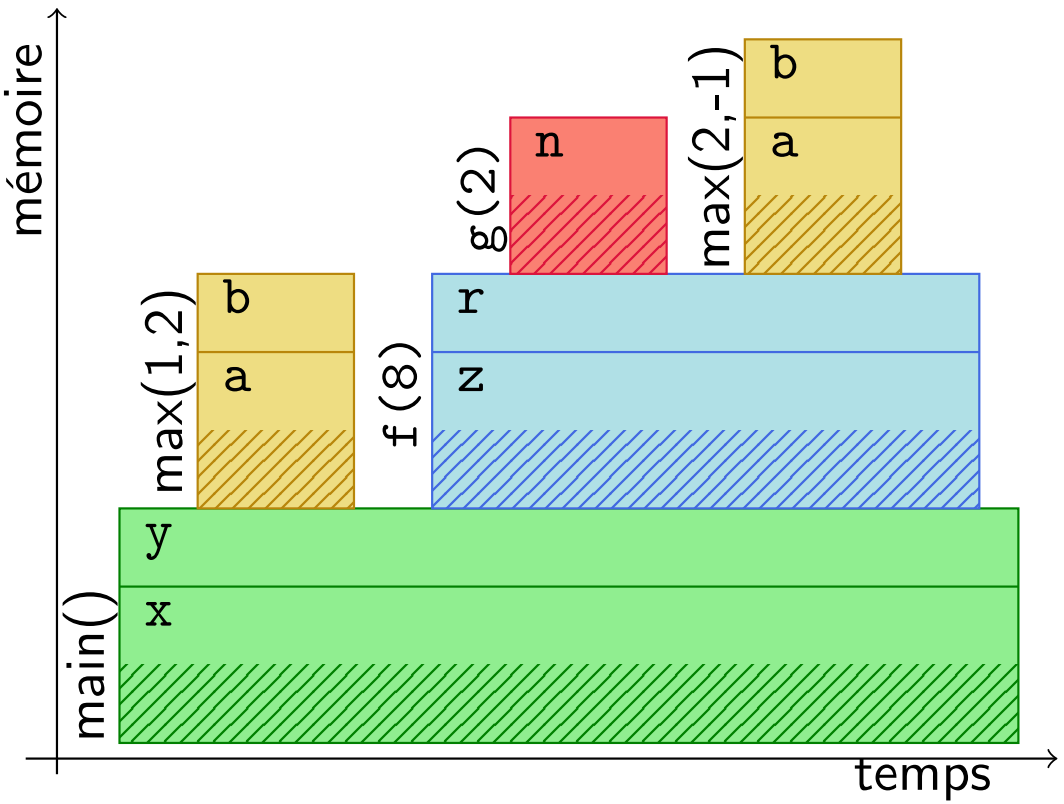
On parle de pile d'appel car les appels de fonctions s'empilent...
comme sur une une pile d'assiettes !



Pile d'appel

On parle de pile d'appel car les appels de fonctions s'empilent...

comme sur une une pile d'assiettes !

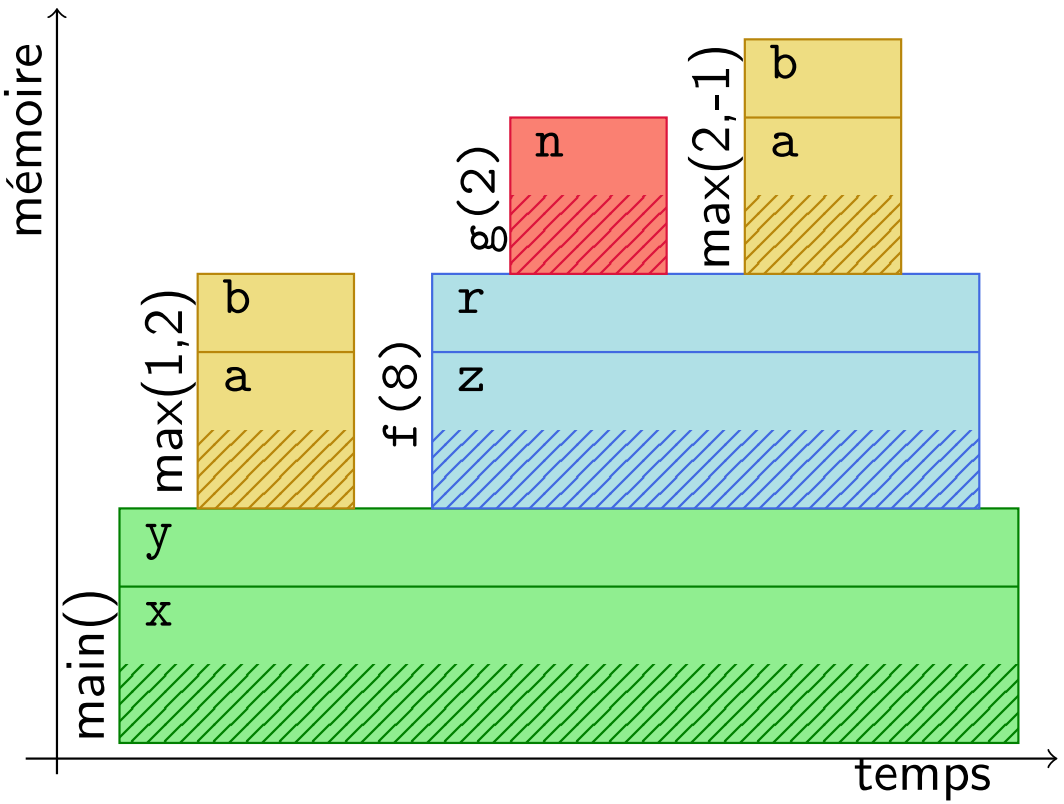


Pile d'appel

On parle de pile d'appel car les appels de fonctions s'empilent...

comme sur une une pile d'assiettes !

Peut-on avoir deux assiettes de la même couleur dans la pile ?



Fonctions récursives

Définition

Une fonction récursive est une fonction dont la définition fait appel à la fonction *elle-même*.

Fonctions récursives

Définition

Une fonction récursive est une fonction dont la définition fait appel à la fonction *elle-même*.

Il y a une forte analogie avec les maths : $(n+1)! = (n+1) \times n!$

Fonctions récursives

Définition

Une fonction récursive est une fonction dont la définition fait appel à la fonction *elle-même*.

Il y a une forte analogie avec les maths : $(n+1)! = (n+1) \times n!$

Terminaison

Il faut un cas de base qui ne déclenche pas d'appel récursif.

Fonctions récursives

Définition

Une fonction récursive est une fonction dont la définition fait appel à la fonction *elle-même*.

Il y a une forte analogie avec les maths : $(n+1)! = (n+1) \times n!$

Terminaison

Il faut un cas de base qui ne déclenche pas d'appel récursif.

Comme dans :

$$f : \mathbb{N} \to \mathbb{N}$$

$$n \mapsto \begin{cases} n \times f(n-1) \text{ si n} > 0 \\ 1 \text{ sinon} \end{cases}$$

Factorielle (1)

```
int factorielle(int n)
{
    int res; /* resultat */
    if (n > 0) /* cas recursif */
    {
        res = n * factorielle(n - 1);
    }
    else /* cas de base */
    {
        res = 1;
    }
    return res;
}
```

Factorielle (2)

Version plus concise :

```
int factorielle(int n)
{
    if (n < 2) /* cas de base */
    {
        return 1;
    }
    return n * factorielle(n - 1);
}
```

Récursion (2). Pour aller plus loin

- Outre les exemples mathématiques directs comme factorielle, de nombreux problèmes sont beaucoup plus facile à résoudre de manière récursive. À au moins un moment du raisonnement, on suppose que l'on dispose déjà de la fonction qui résout le problème et on l'applique à un cas « plus petit ».

Récursion (2). Pour aller plus loin

- Outre les exemples mathématiques directs comme factorielle, de nombreux problèmes sont beaucoup plus facile à résoudre de manière récursive. À au moins un moment du raisonnement, on suppose que l'on dispose déjà de la fonction qui résout le problème et on l'applique à un cas « plus petit ».
- On apprend ici la programmation impérative où un élément central est le changement d'état des cases mémoires (et les effets de bord comme vous le verrez au second semestre). En *programmation fonctionnelle*, l'accent est mis sur les fonctions sans effets de bord, et la récursion occupe le tout premier plan, notamment pour faire ce que l'on a l'habitude de faire avec des boucles en programmation impérative.

Récursion (2). Pour aller plus loin

- Outre les exemples mathématiques directs comme factorielle, de nombreux problèmes sont beaucoup plus facile à résoudre de manière récursive. À au moins un moment du raisonnement, on suppose que l'on dispose déjà de la fonction qui résout le problème et on l'applique à un cas « plus petit ».
- On apprend ici la programmation impérative où un élément central est le changement d'état des cases mémoires (et les effets de bord comme vous le verrez au second semestre). En *programmation fonctionnelle*, l'accent est mis sur les fonctions sans effets de bord, et la récursion occupe le tout premier plan, notamment pour faire ce que l'on a l'habitude de faire avec des boucles en programmation impérative.
- Un appel récursif peut être indirect, c'est à dire effectué dans le code d'une fonction auxilliaire.

Exemples. Affichage à la descente

Avec le code :

```
int factorielle(int n)
{
    printf("%d ", n);
    if (n < 2) /* cas de base */
    {
        return 1;
    }
    return n * factorielle(n - 1);
}
```

L'appel `factorielle(5)` aura pour effet de bord d'afficher :

Exemples. Affichage à la descente

Avec le code :

```
int factorielle(int n)
{
    printf("%d ", n);
    if (n < 2) /* cas de base */
    {
        return 1;
    }
    return n * factorielle(n - 1);
}
```

L'appel `factorielle(5)` aura pour effet de bord d'afficher :

5 4 3 2 1

Exemples. Affichage à la descente

Avec le code :

```
int factorielle(int n)
{
    printf("%d ", n);
    if (n < 2) /* cas de base */
    {
        return 1;
    }
    return n * factorielle(n - 1);
}
```

L'appel `factorielle(5)` aura pour effet de bord d'afficher :

`5 4 3 2 1`

- Comment obtenir `1 2 3 4 5` ?

Exemples. Affichage à la remontée

Et avec le code :

```
int factorielle(int n)
{
    int res = 1;
    if (n > 1) /* cas recursif */
    {
        res =  n * factorielle(n - 1);
    }
    printf("%d␣", n);
    return res;
}
```

L'appel `factorielle(5)` aura pour effet de bord d'afficher :

Exemples. Affichage à la remontée

Et avec le code :

```
int factorielle(int n)
{
    int res = 1;
    if (n > 1) /* cas recursif */
    {
        res =  n * factorielle(n - 1);
    }
    printf("%d ", n);
    return res;
}
```

L'appel `factorielle(5)` aura pour effet de bord d'afficher :

`1 2 3 4 5`

- Comment obtenir `5 4 3 2 1 1 2 3 4 5` ?
- Peut-on obtenir : `1 2 3 4 5 5 4 3 2 1` ?

Exemple. Double appel

Coefficients binomiaux :

$$\binom{n}{p} = \frac{n!}{p!(n-p)!}$$

Exemple. Double appel

Coefficients binomiaux :

$$\binom{n}{p} = \frac{n!}{p!(n-p)!}$$

Relation de récurrence donnée par le triangle de Pascal :

$$\binom{n+1}{p+1} = \binom{n}{p} + \binom{n}{p+1}$$

Exemple. Double appel

Coefficients binomiaux :

$$\binom{n}{p} = \frac{n!}{p!(n-p)!}$$

Relation de récurrence donnée par le triangle de Pascal :

$$\binom{n+1}{p+1} = \binom{n}{p} + \binom{n}{p+1}$$

Cas de base : $\binom{n}{0} = \binom{n}{n} = 1$

Exemple. Double appel

Coefficients binomiaux :

$$\binom{n}{p} = \frac{n!}{p!(n-p)!}$$

Relation de récurrence donnée par le triangle de Pascal :

$$\binom{n+1}{p+1} = \binom{n}{p} + \binom{n}{p+1}$$

Cas de base : $\binom{n}{0} = \binom{n}{n} = 1$

Code :

```
int binomial(int n, int p)
{
    if ( (p == 0) || (n == p) ) /* cas de base */
    {
        return 1;
    }
    return binomial(n - 1, p - 1) + binomial(n - 1, p);
}
```

Exemple. Écriture récursive de boucles

Calcul de la moyenne d'une série saisie par l'utilisateur.

Exemple. Écriture récursive de boucles

Calcul de la moyenne d'une série saisie par l'utilisateur.

```
double faire_moyenne()
{
    return faire_moyenne_aux(0, 0);
}

double faire_moyenne_aux(double somme, int n)
{
    int terme = -1;

    printf("Entier positif : ");
    scanf("%d", &terme);
    if (terme < 0) /* cas de base */
    {
        return somme / n; /* moyenne des termes precedents */
    }
    return faire_moyenne_aux(somme + terme, n + 1);
}
```